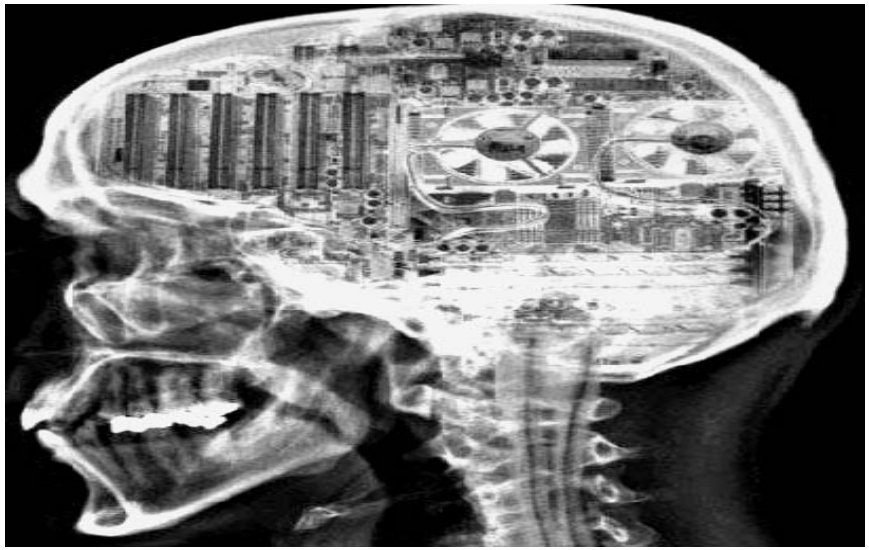
UNIVERSITÉ PARIS-SUD 11

Licence Mention Informatique – L2/S4 – 2011

Programmation Object & Genie Logiciel

Case-Study: Attol



*Frederic Voisin - **Burkhart Wolff***
Département Informatique

Plan of the Chapter

- Requirements of ATTOL (Cahier de Charges)
- Using UML in Analysis (Analyse)
  - Use-Cases
  - Sequence Charts
  - State Machines
  - Class Diagrams
- Using UML in Design (Conception)

En guise de cahier des charges…

- Une **station service** *ATTOL* a plusieurs **postes** de distribution
  - soit *automatiques* (par carte bancaire) ouverts 24h/24
  - soit *manuels* (utilisables seulement si la station est ouverte)

  Chaque poste est identifié par un numéro.
- Chaque poste peut délivrer 3 sortes de carburant. A chaque instant, il en délivre au plus une sorte.
- Chaque poste est muni de 3 compteurs
  - la quantité de carburant servie
  - le prix au litre
  - le prix à payer (qui ne change pas pendant la journée)

  Les compteurs affichent 0 s'il n'y a pas de distribution en cours.
- La station dispose de 3 **citernes** (une par type de carburant) avec
  - le niveau courant de carburant,
  - un niveau d'alerte pour prévenir le gérant qu'elle va être vide
  - un niveau minimal qui, une fois atteint, ne permet plus de délivrer du carburant.

  Les niveaux d'alerte et minimaux sont les mêmes pour toutes les citernes.
- Le système doit garder une trace de tous les **achats** effectués depuis la dernière ouverture de la station.

En guise de cahier des charges… (2)

- Un opérateur **ouvre** la station quand il arrive et la **ferme** quand il part
- Pour **utiliser un poste automatique** :
  - Un client insère sa carte bancaire et tape son code
  - Le système authentifie la carte auprès du service bancaire
  - En cas de succès, le client choisit un type de carburant et se sert d'un certain nombre de litres.
  - Le système enregistre l'achat, envoie un ordre de débit au service bancaire et remet à zéro les compteurs du poste.
- Pour utiliser un **poste manuel** :
  - le client choisit son carburant et se sert.
  - Les caractéristiques de l'achat sont envoyées à l'opérateur.
  - Lorsque le client a payé en indiquant son numéro de pompe, le pompiste **enregistre** l'achat et remet à zéro les compteurs du poste

Les postes restent bloqués tant que les compteurs n'ont pas été remis à 0.

- Quand une **livraison de carburant** a eu lieu, l'opérateur met à jour le niveau de la citerne. Une livraison fait toujours repasser au-dessus de l'alarme.

Limitations de (la manière de traiter) l'exemple

- On ne modélise pas le début et la fin de la distribution, ni la mise à jour des compteurs pendant la distribution
- On ne décrit pas comment le client choisit le carburant, ni l'authentification de la carte bancaire
- Il n'y a pas de « simultanéité » d'évènements
- On suppose que le niveau minimal est suffisant pour assurer toute distribution commencée
- Le réapprovisionnement livre toujours assez de carburant
- …

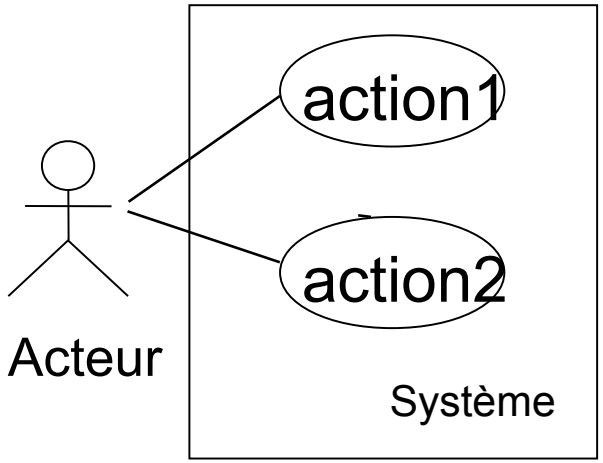
☞ On modélise en supposant qu'on connaît au début de l'opération le volume et le type de carburant acheté!
La distribution apparaît ici comme « atomique » et non pas continue.

Les « cas d'utilisation »

- les frontières du système
- les fonctionnalités attendues
- les utilisateurs et les partenaires

Les « cas d'utilisation » en UML 2

- Un « cas d'utilisation » :
  - une **fonctionnalité** du système
  - déclenchée par un **acteur extérieur**
- **Acteur** : tiers qui joue un rôle
  - En interagissant avec le système :
    Il envoie des données / signaux au système et/ou en reçoit des informations
  - Il est extérieur au système



- *Dans le système, on a parfois une **représentation** de l'acteur (ex. informations sur le client)*

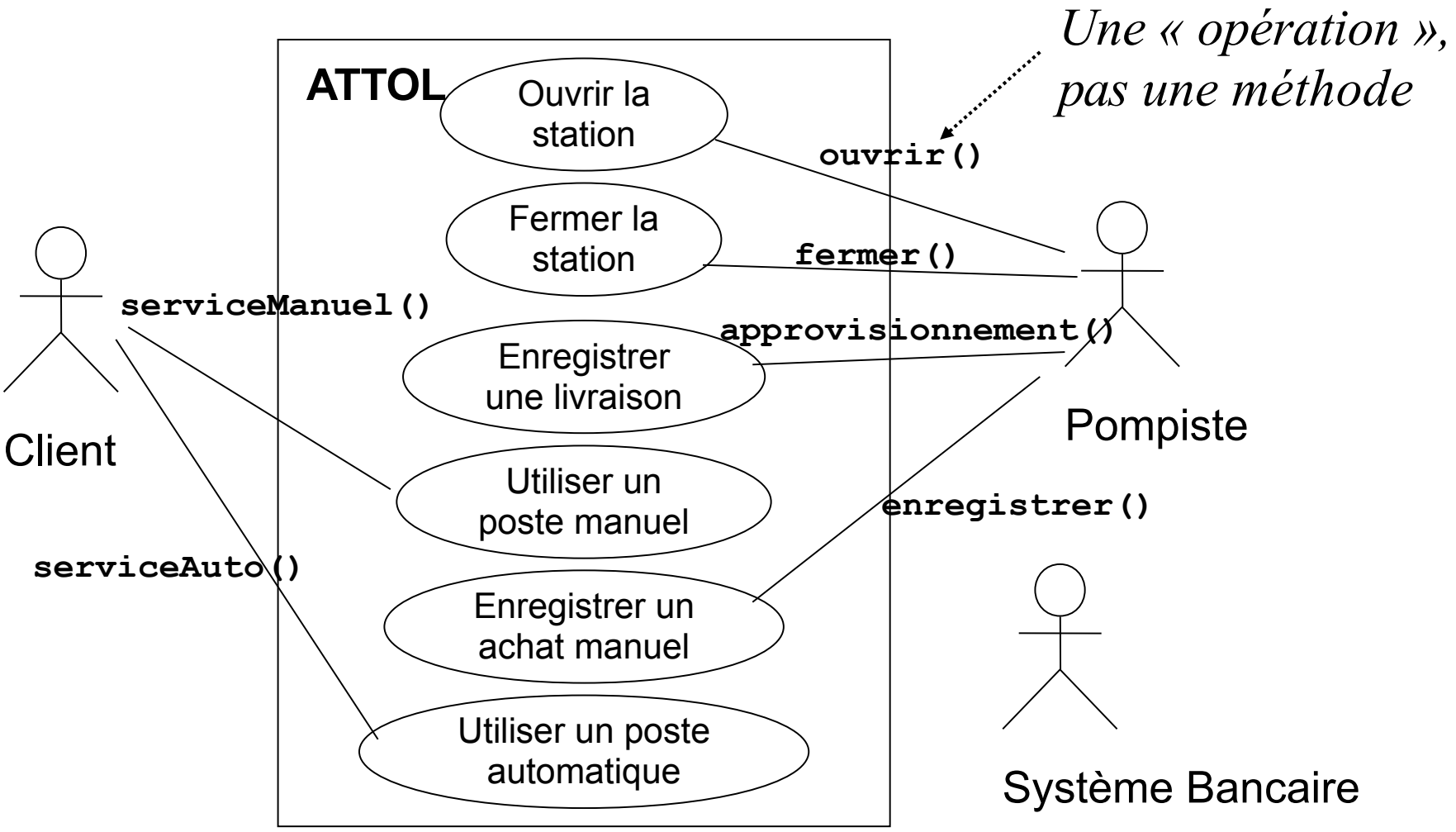
- *Une même personne peut jouer le rôle de plusieurs acteurs (opérateur, client) dans le temps.*
- *Plusieurs personnes peuvent avoir le même rôle: vu du système il s'agit du même acteur (les « instances » d'un acteur sont anonymes)*

Un acteur est une « personne logique »

Les cas d'utilisation en UML 2 (2)

- Les cas d'utilisation décrivent le **comportement** du système, les interactions qu'il a avec l'extérieur.
- Un système ne fait pas des choses spontanément, mais en réaction à une sollicitation initiale par un « acteur » extérieur
- On doit préciser un cas d'utilisation
  - en décrivant les flots d'événements à l'aide d'un **texte** ;
  - À l'aide de **diagrammes de séquences** pour préciser graphiquement ces flots.
- Il faut prendre en compte
  - Les cas normaux, leurs variantes éventuelles
  - Les cas « d'erreurs »
- Il faudra aussi mettre en évidence les interactions **entre** fonctionnalités des cas d'utilisation (diagrammes de séquence)

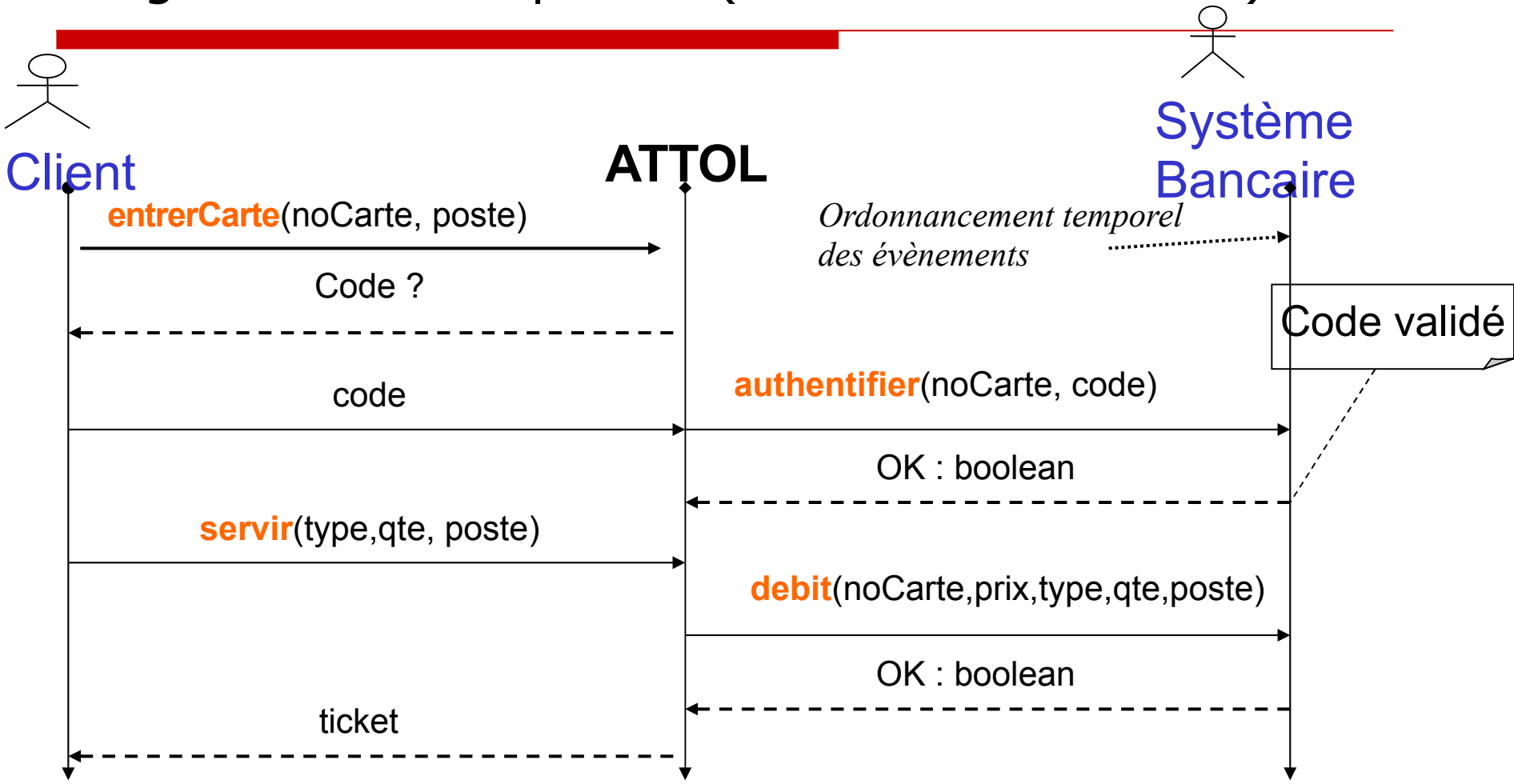
Diagramme des cas d'utilisation ATOLL



Diagrammes de séquence **en analyse**

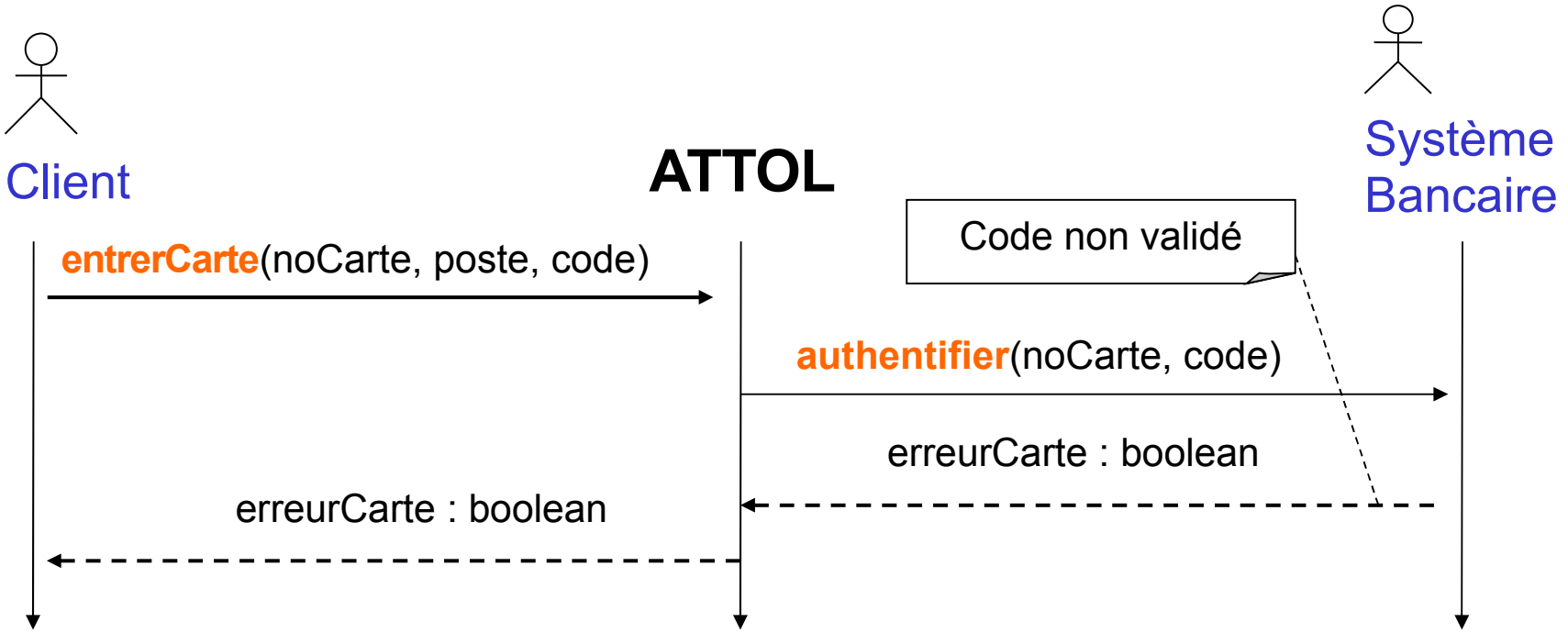
- Ils correspondent à des « diagrammes système »
  - le système est vu comme une « boite noire »
  - Échanges de messages entre les acteurs et le système
    - ☞ jamais entre acteurs (pas du ressort du système !)
    - ☞ ni entre éléments internes au système (conception)
  - Représentation de la chronologie des échanges
  - Pas de boucle, branchement !
    - Un chemin principal, des alternatives,
    - des cas d'exception, des commentaires
- Usages :
  - Illustrer **chaque cas isolément** (diagramme très simple)
  - Illustrer les **interactions entre cas** d'utilisation (scenarii)
- D'autres types de diagrammes de séquence seront vus en **conception** pour illustrer le **déroulement interne** d'une opération

Diagramme de séquence (détails d'interaction)



« utiliser un poste automatique » : cas principal

Diagramme de séquence d'un cas d'utilisation



« utiliser un poste automatique » (2)

- Erreurs ou signaux renvoyés
- Ici, on a fait le choix de ne pas détailler les interactions

Exemple de synthèse d'un cas d'utilisation

Utiliser un poste automatique	
Cas principal	Un client utilise un poste automatique après avoir authentifié sa carte bancaire et sélectionné un carburant. Le niveau de ce carburant reste au-dessus du niveau d'alerte .
Cas alternatif	Après utilisation d'un poste automatique par un client, le seuil d'alarme du carburant est franchi.
	Après utilisation d'un poste automatique par un client, le seuil minimal du carburant est franchi.
Cas d'exceptions	Echec d'authentification de la carte bancaire d'un client d'un poste automatique
Remarques	L'interface du système garantit qu'en cas de niveau de carburant inférieur au minimum, la sélection du carburant est impossible sur tous les postes. Ce niveau est tel que toute distribution commencée peut être achevée correctement

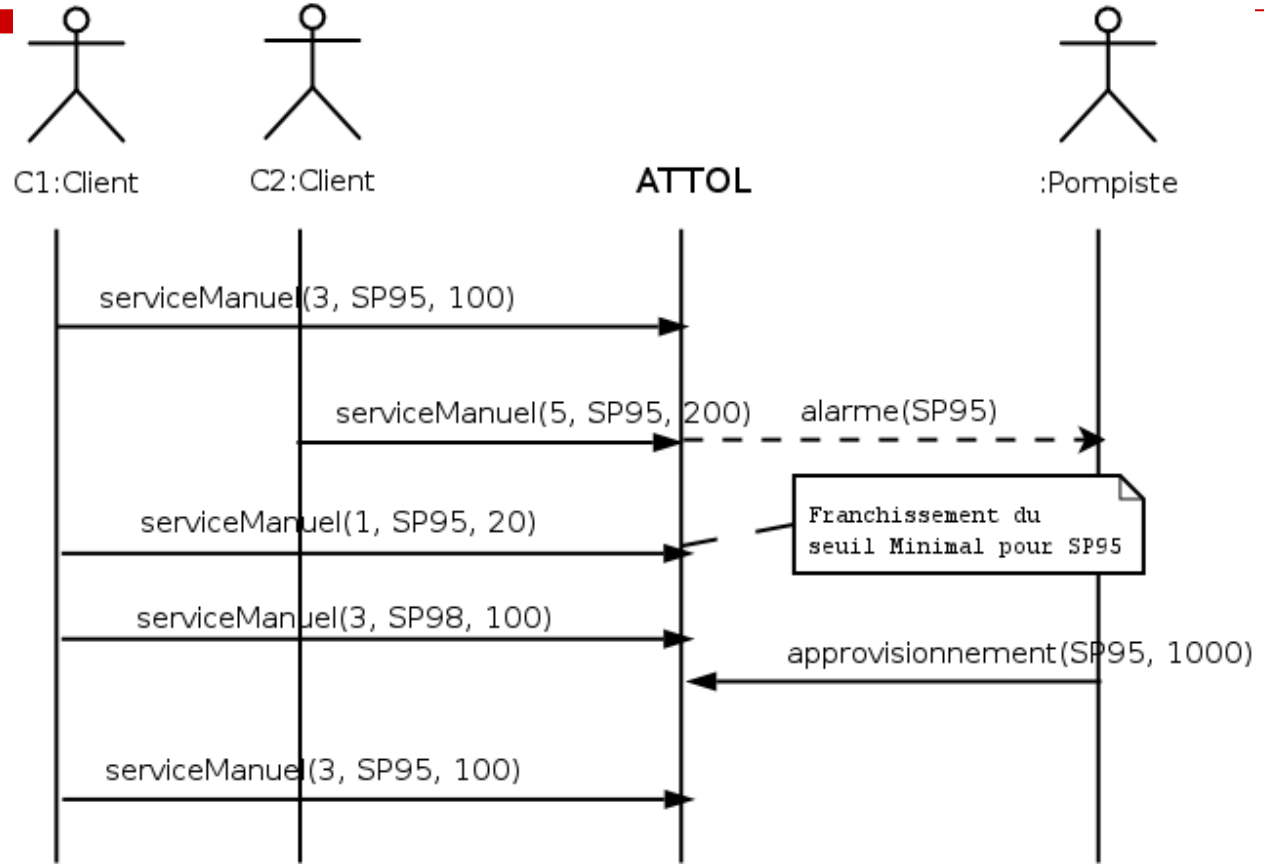
Description inspirée de « Developing Applications with Java and UML », P. R. Reed, Addison-Wesley

Description textuelle d'un cas d'utilisation

Nom	*nom de l'opération*
Objectif	*en une ou deux phrases claires*
Acteur principal	*initiateur de l'opération*
Acteurs secondaires	*autres acteurs impliqués*
Pour chaque cas d'utilisation…	
Statut	*(principal, alternatif, d'exceptions)*
Etapes principales	*Enumération des étapes principales et des conditions de déclenchement*
pour chaque extension…	
Extension	nom de l'extension
	point de rattachement de l'extension
	Etapes principales de l'extension

Description inspirée de « Developing Applications with Java and UML », P. R. Reed, Addison-Wesley

Un diagramme de séquence **inter** cas d'utilisation



*Ce scénario n'est pertinent que s'il permet d'illustrer **un** fonctionnement global du système. Quels sont les « évènements » mis en valeur ?*

Retour sur le diagramme précédent

- Ce qu'on arrive à exprimer sur cet exemple :
  - Les carburants ne sont pas associés à un poste précis
  - Si le niveau est compris entre le niveau d'alarme et le niveau minimal, on peut continuer à se servir de ce carburant

  …

  Penser aux scenarii pertinents, qui illustrent l'usage attendu du système, l'interaction entre opérations

- Ce qu'on n'arrive **pas** à exprimer :

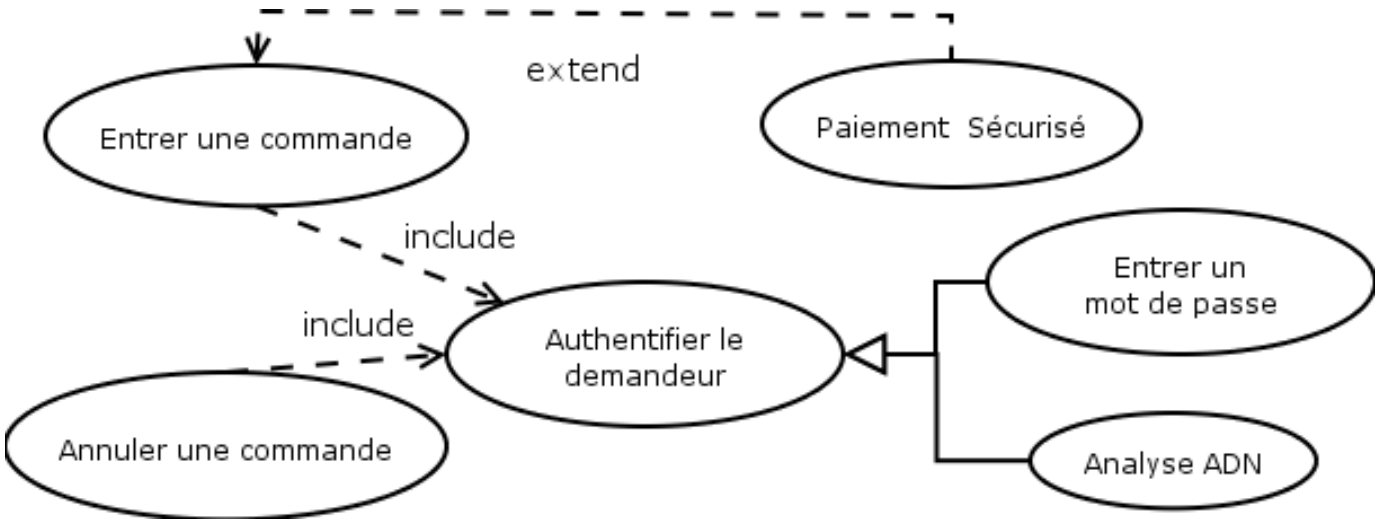
  Quand le niveau minimal est atteint, on ne peut plus se servir d'un carburant : l'opération est inhibée (non exprimable !) et il n'y a pas « d'observateur » du niveau d'une citerne.

  On ne peut pas illustrer ce cas par un scénario (et on ne pourra pas le tester !)

☞ Identification d'un manque « d'observateurs » dans ce système !

Relations entre cas d'utilisation

A utiliser
avec modération !



extend:

- Le cas de base a prévu une liste de points d'extensions, ainsi que des conditions d'activation

include:

- Le cas d'utilisation n'existe pas isolément
- Le cas de base mentionne explicitement les points d'inclusion

généralisation: on évitera, sauf si cela apporte vraiment quelque chose

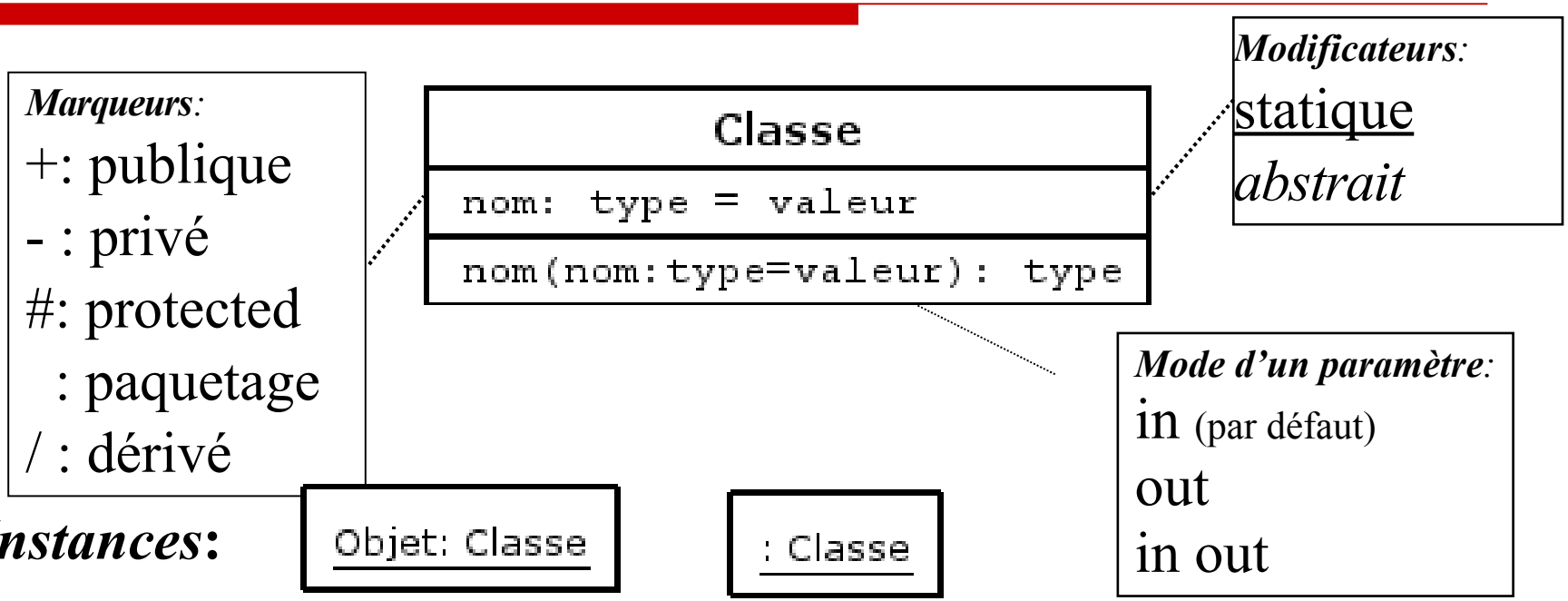
Quelques règles sur les cas d'utilisation

- Entre 3 et 6 fonctionnalités par diagrammes de cas d'utilisation. Si plus, introduire un cas d'utilisation plus abstrait qu'on raffinera ensuite dans un nouveau diagramme
- Il est plus utile d'avoir une bonne description textuelle qu'un diagramme sophistiqué avec de nombreuses relations
- Illustrez votre diagramme par des scénarii pertinents illustrant les aspects dynamiques du système, les acteurs impliqués, les cas d'erreur.

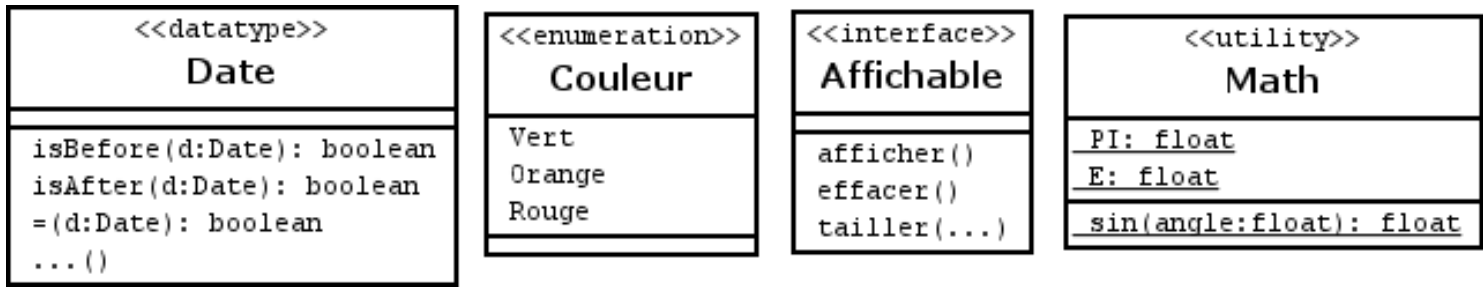
Le diagramme de classes :

- les entités concernées
- leurs relations

Syntaxe des Classes et Instances en UML 2



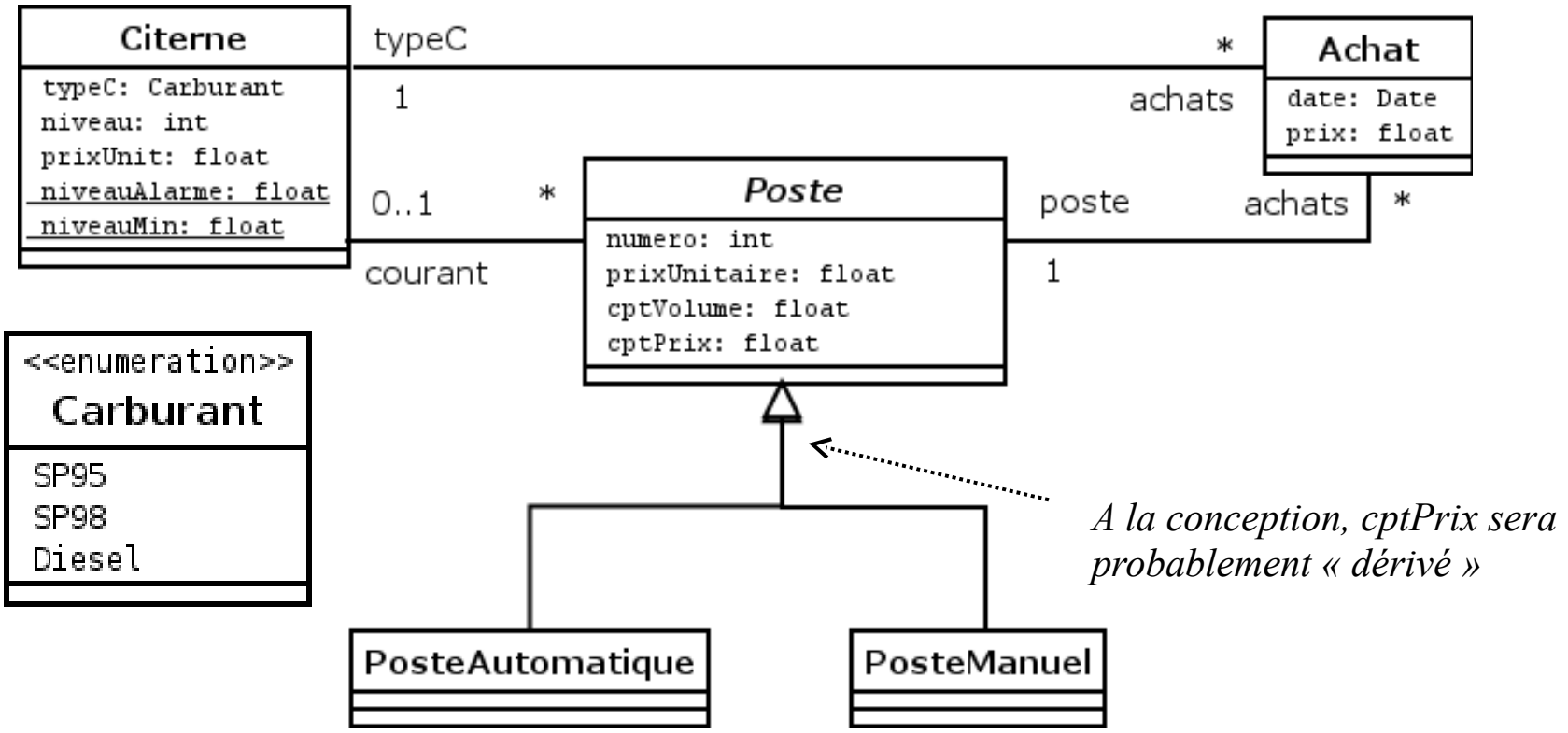
Stéréotypes:



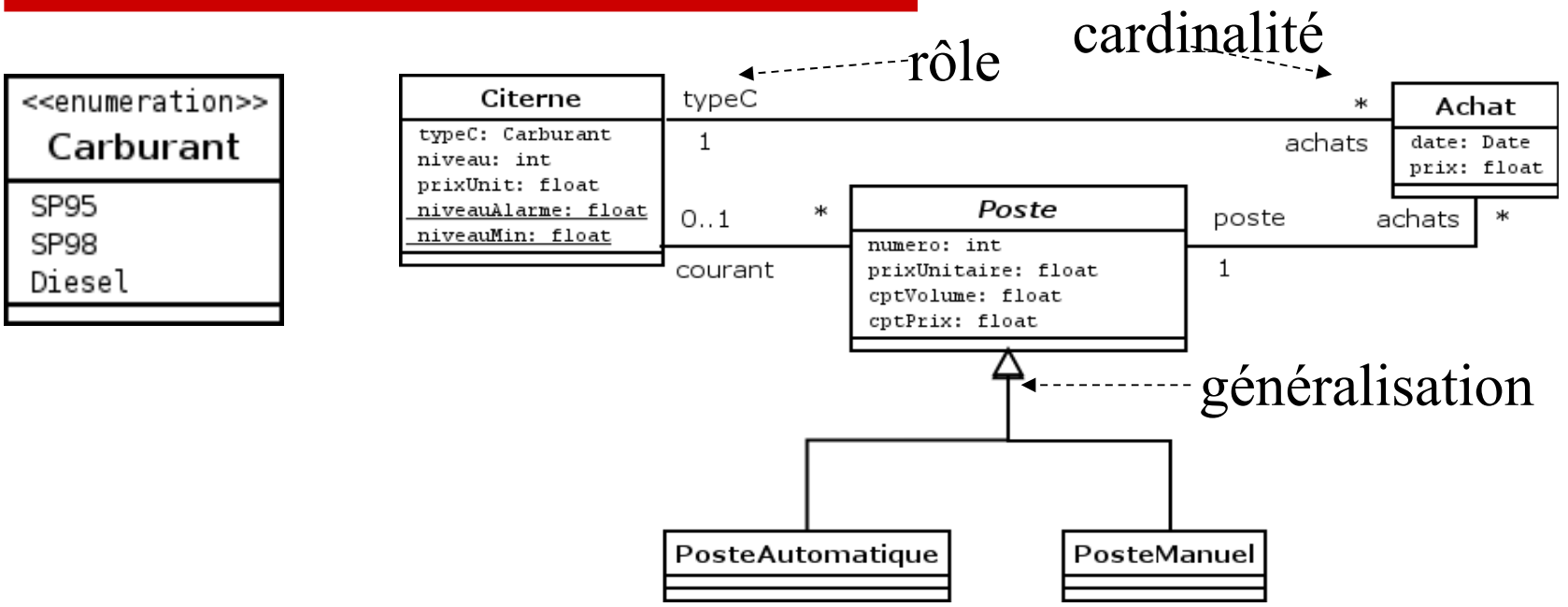
Les classes en phase d'analyse

- Les attributs
  - De type simple (int, boolean, …) ou primitif (Date) uniquement !
  - Les liens avec les autres classes d'intérêt sont représentés par les **associations**
  - Dans le modèle d'analyse, on les considère comme **public** (on raffinera à la conception, ainsi que les méthodes d'accès)
  - On ne distingue pas les attributs « dérivés » des autres
- Les opérations
  - dans un premier temps, on ne considère que les opérations principales (liées aux cas d'utilisation)
  - on ne les rattache pas à une classe particulière, donc on ne se préoccupe pas des méthodes à ce niveau (quels paramètres ?)

Une première version



Les associations en UML 2



Pour `a: Achat`, `a.poste` correspond à **une** instance de `Poste`.

Pour `c: Citerne`, `c.achats` correspond à une **collection** d'instances de `Achat`

Pour `p: Poste`, `p.courant` correspond à une **collection de 0 ou 1** `Citerne`.

Les rôles permettent de naviguer à travers les associations

Le nom de classe peut servir de rôle par défaut (si pas d'ambiguïté)

Cardinalité/Multiplicité

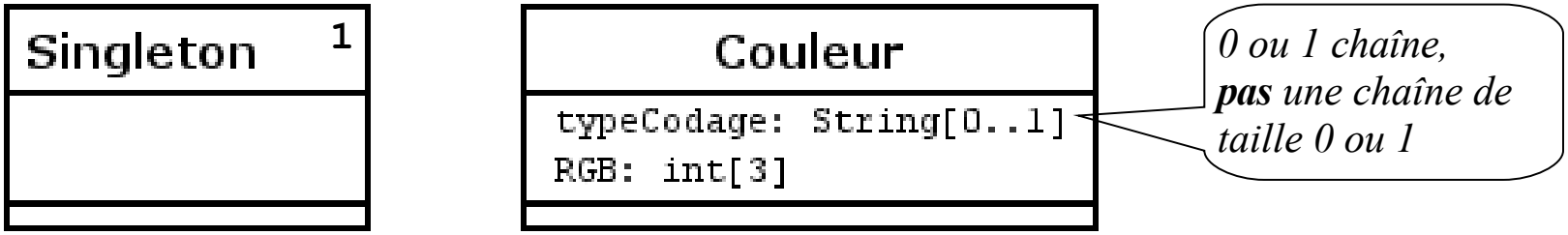
Cardinalités dans les associations: de la forme

- 1, 2, ou un nombre entier précis (**pas** une expression !)
- * : un nombre quelconque, éventuellement nul
- un intervalle comme 1..*, 0..1, 1..3, (**pas** 1..N)
  - *on donnera systématiquement les cardinalités*
  - *Attention à la différence: une instance (1), au plus une instance (0..1), une collection d'instances (* ou 1..*),*

Sémantique d'une association *: des tuples*

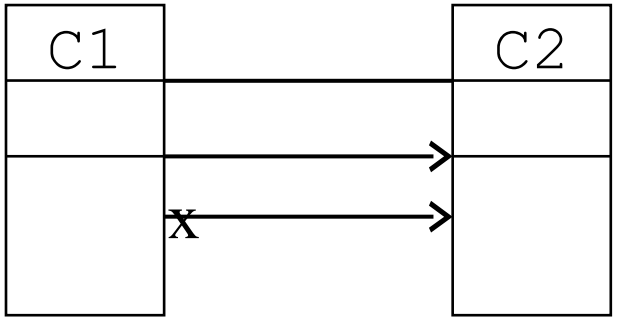
On peut associer une multiplicité aux attributs et classes

- *À utiliser avec précaution !*



Traversée d'associations

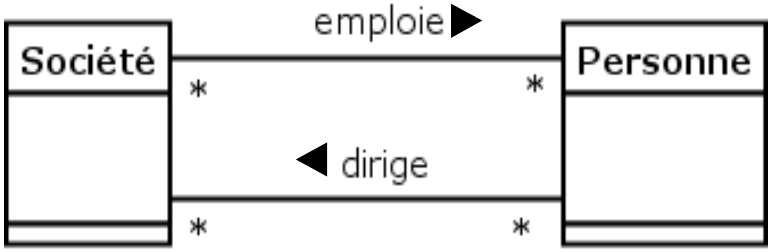
Navigabilité: 3 possibilités



- Dans la phase d'analyse, on se limite à des associations implicitement navigables dans les deux sens
- A la conception, on pourra choisir de ne conserver qu'un sens
- En pratique, la navigabilité « réelle » se verra en fonction des rôles nécessaires pour exprimer les invariants

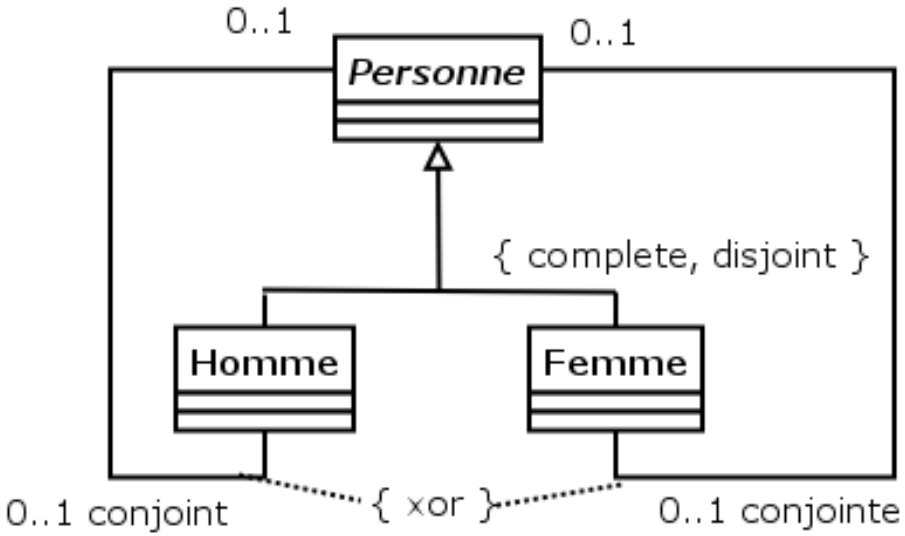
Nommage : on peut nommer une association

- *on privilégie plutôt les noms de rôles…*
- *on donnera un rôle si on en a besoin dans une contrainte*

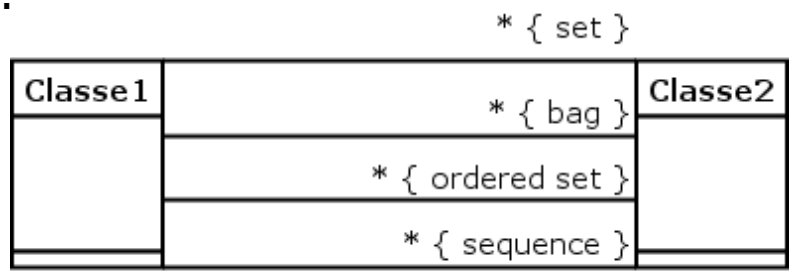


Contraintes posées sur une association

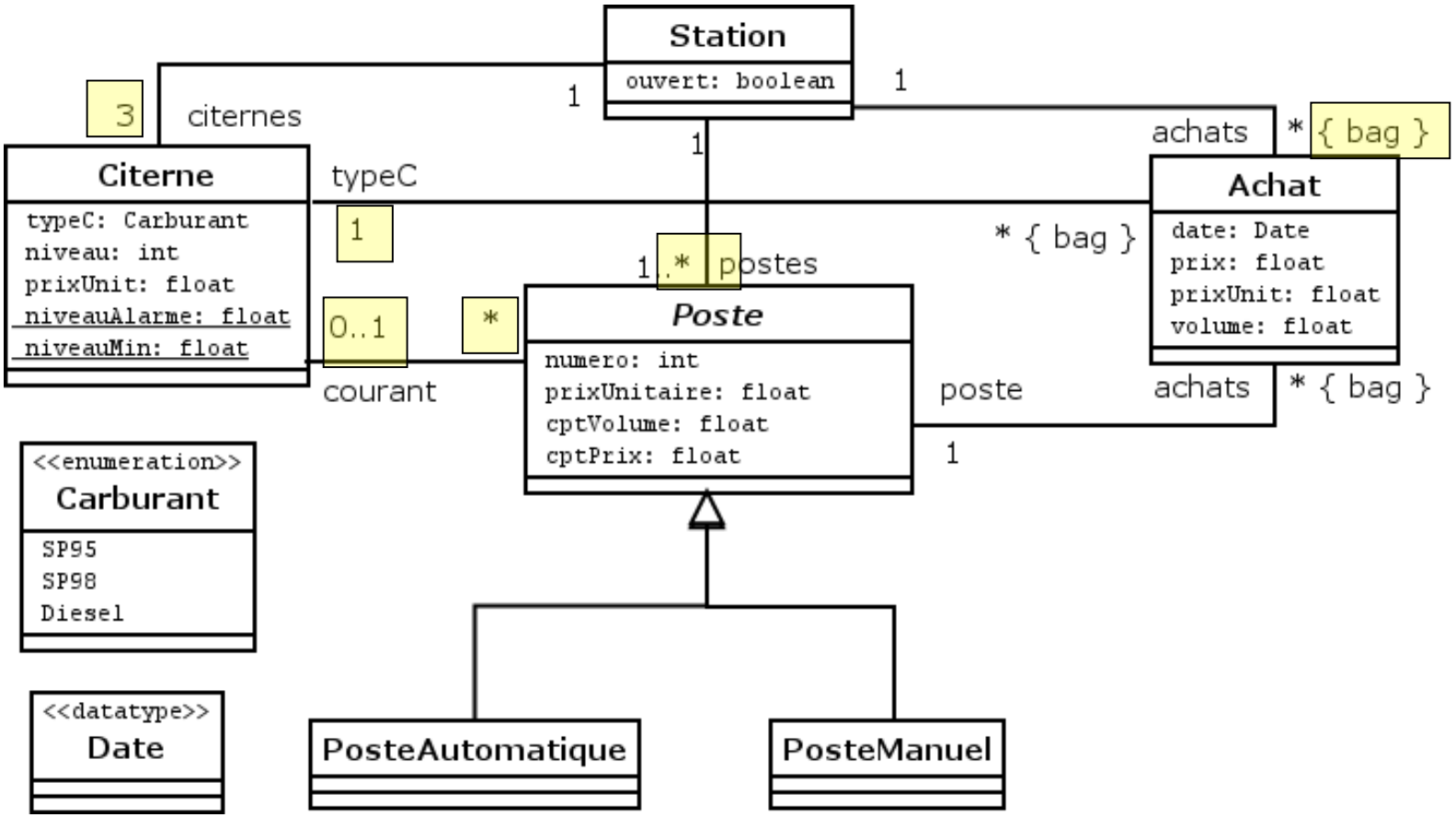
- Pour la généralisation:
  - `complete`, `incomplete`
  - `disjoint`, `overlapping`
- Entre associations
  - `xor` (souvent ambigu, on lui préfèrera une contrainte explicite)
- Immuabilité d'une extrémité de lien (`readonly`)



- Contraintes associées à la multiplicité `*`:
  - Ordonnée ou non?
  - Avec duplication des éléments ?

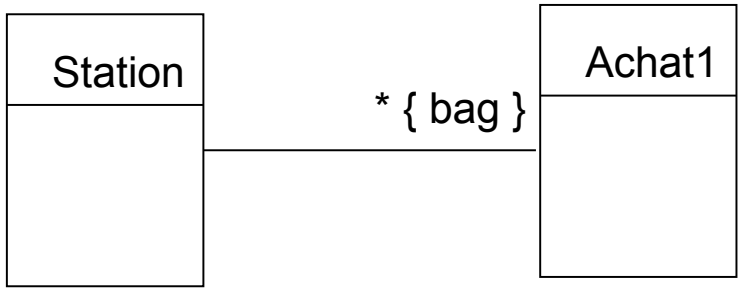


Les associations pour ATTOL

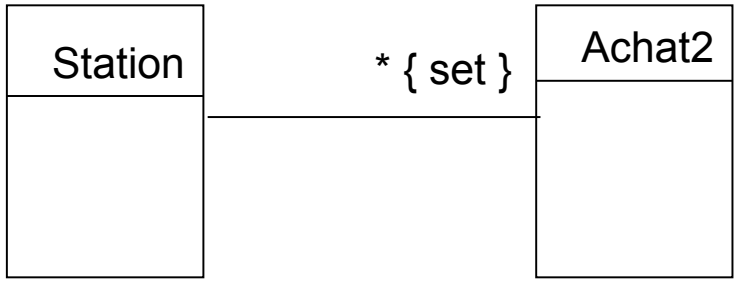


Bag ou Set: vaut mieux y regarder à deux fois…

```
public class Achat1 {
public boolean equals(Object o) {
    if (o == null) return false;
    try { Achat1 a = (Achat1) o;
          return a.date.equals(date)
               && a.prix == prix && …
    } catch (ClassCastException e)
      { return false; }
}
```




```
public class Achat2  {
  public boolean equals(Object o) {
     return this == o;
  }
}
```



Vous préférez un bag{Achat1} ou un set{Achat2} ?

Ça dépend aussi de la précision associée à la notion de Date !

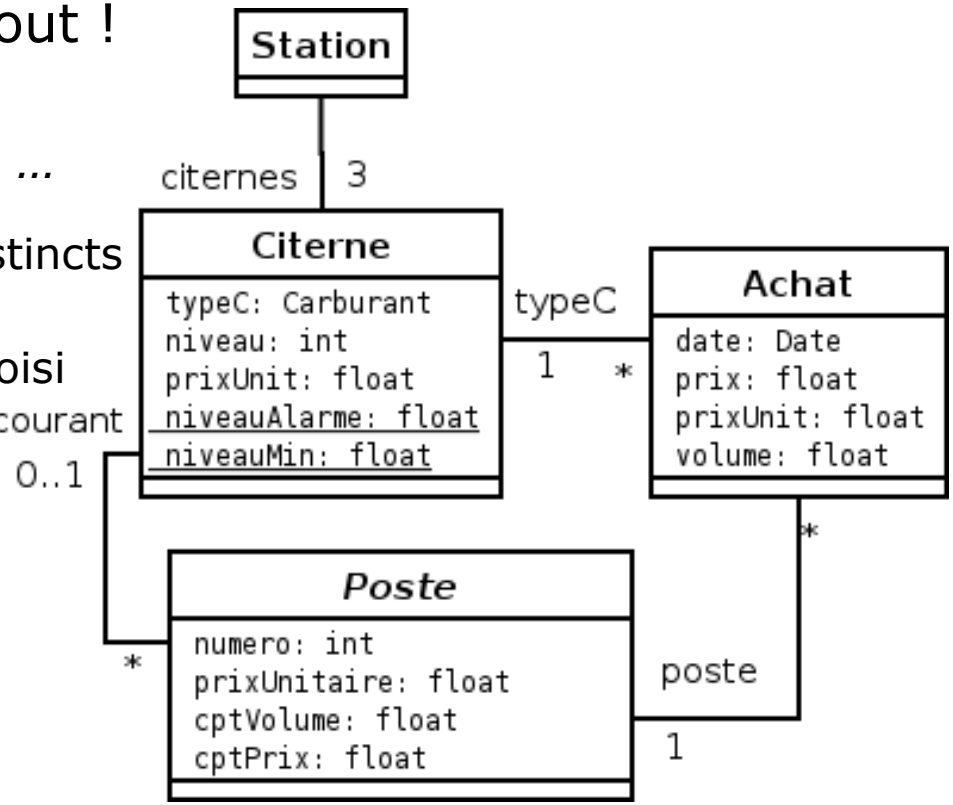
ATTOL: au-delà du diagramme de classes

Le diagramme n'exprime pas tout !

☞ *Nécessité d'invariants informels …*

- les citernes ont des carburants distincts
- en cours de distribution le prix affiché est le prix du carburant choisi
- s'il n'y a pas de carburant choisi, les compteurs sont à 0
- niveauMin ≤ nivAlarme
- cptPrix = prixUnitaire*cptVolume

…



☞ *ou formels (OCL) !*

```
context p:Poste
inv paiement: p.cptPrix = p.cptVolume * p.prixUnitaire
inv RAZ: p.courant->empty() implies
                          p.cptVolume = 0 and p.prixUnitaire = 0
```

ATTOL: redondance d'associations

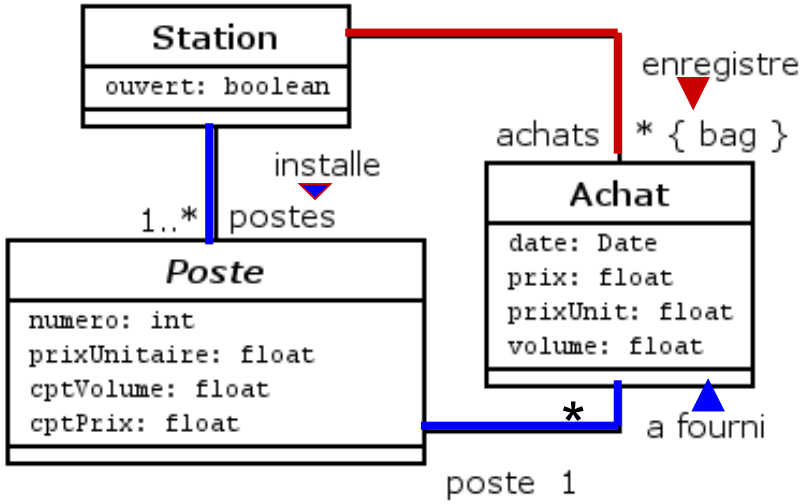
Il existe plusieurs moyens d'accéder aux achats !

- Par navigation avec **`installe`** et « **`a fourni`** »
- via **`enregistre`**

Sont-elles cohérentes ?

Invariant ?

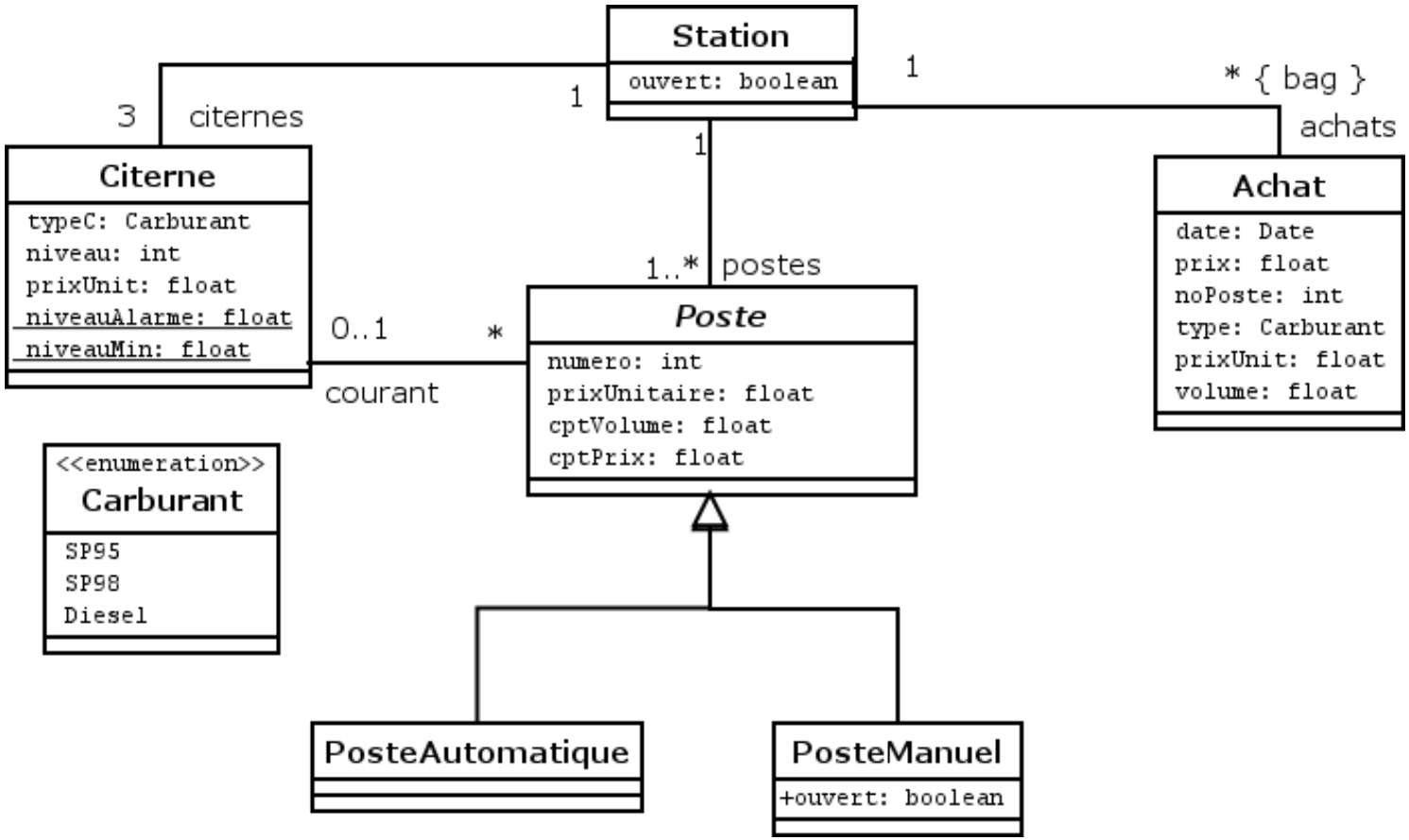
« la collection d'achats **`enregistre`** est la réunion, sur l'ensemble des postes **`installe`** de la station, des collections d'achats **`a fourni`** de chaque poste»



A l'analyse : il faut se demander si on garde la redondance

A la conception : Idem ! Ou comment navigue-t-on ?

ATTOL sans association redondante

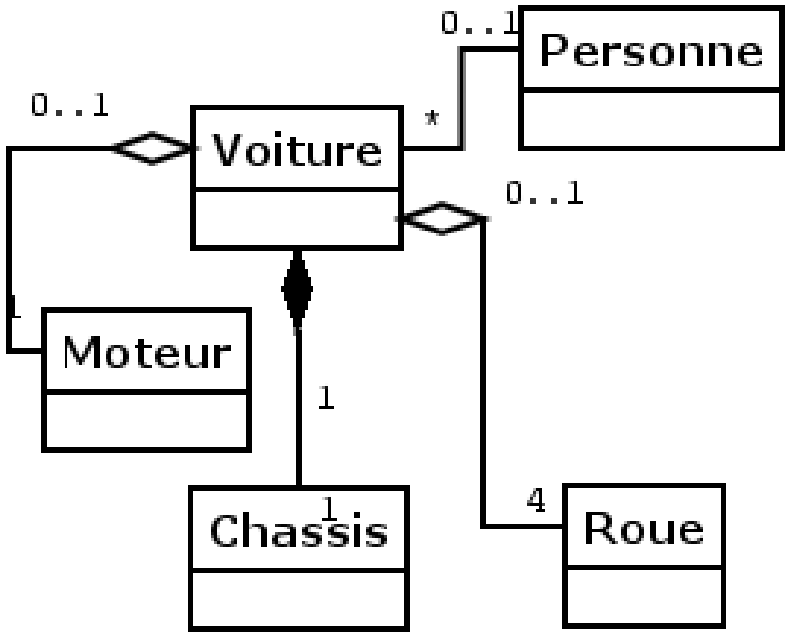


Remplacen
l'associatio

☞ *On a maintenant besoin d'un invariant sur* `noPoste` *!*

Variations sémantiques d'associations

- Association simple
- Agrégation : « est composé de »



- antisymétrie et transitivité des liens
- Associations binaires uniquement !

- Composition : restriction d'agrégation
  - Un composant ne peut pas être partagé
  - La durée de vie du composant est gérée par l'instance propriétaire

- des variations « sémantiques » !
- Java : même représentation pour les 3…
  (C++: Sous-objet ? Pointeur ? Référence ?)

Variations sémantiques d'associations (2)

Généralisation/Spécialisation

- Pas le droit de re-déclarer un attribut
- Les méthodes redéfinies doivent coïncider sémantiquement avec la superclasse sur les aspects communs (pas de redéfinition arbitraire !)
- Pas limité à de l'héritage simple,
- ni à un typage statique (classification dynamique)

☞ Traduction dans un langage de programmation ?

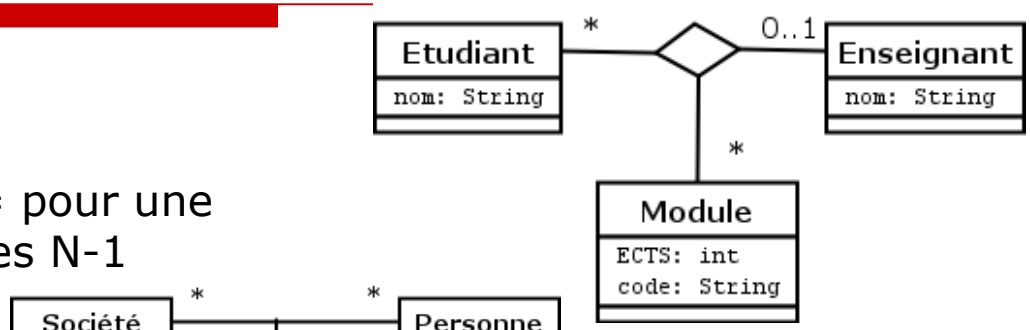
- la sous-classe hérite des contraintes de la superclasse (invariants, pré/post-conditions des méthodes !)

Les associations « exotiques »

Suit le cours

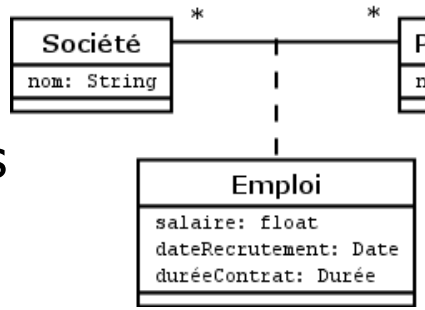
Association N-aire

Multiplicité ? Sur une branche = pour une instance fixée à chacune des N-1 autres branches

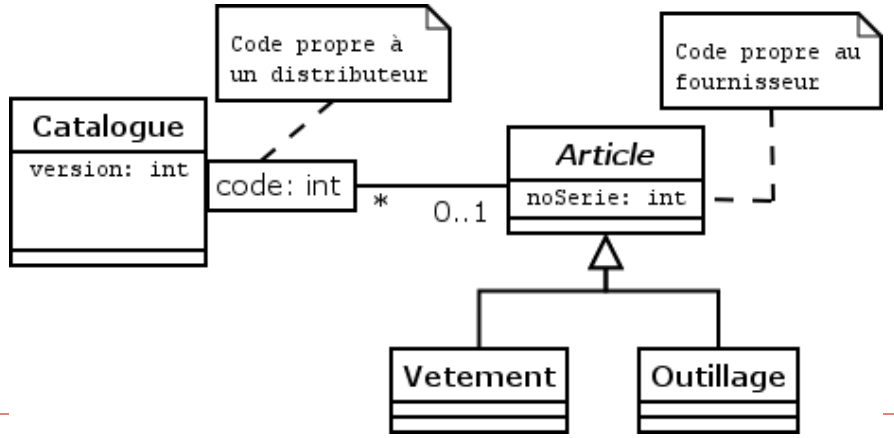


Association avec attributs

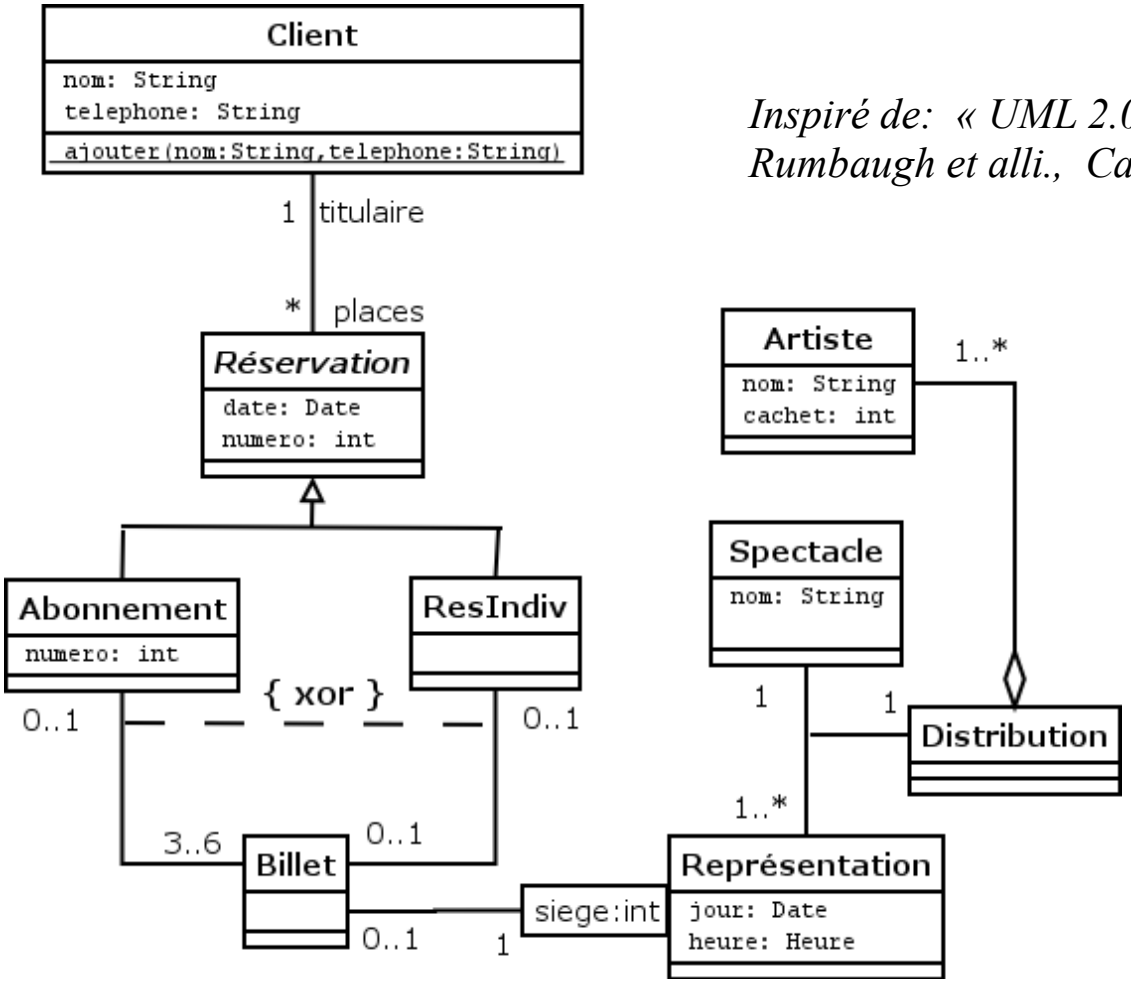
(« classe association »)



Association « qualifiée »

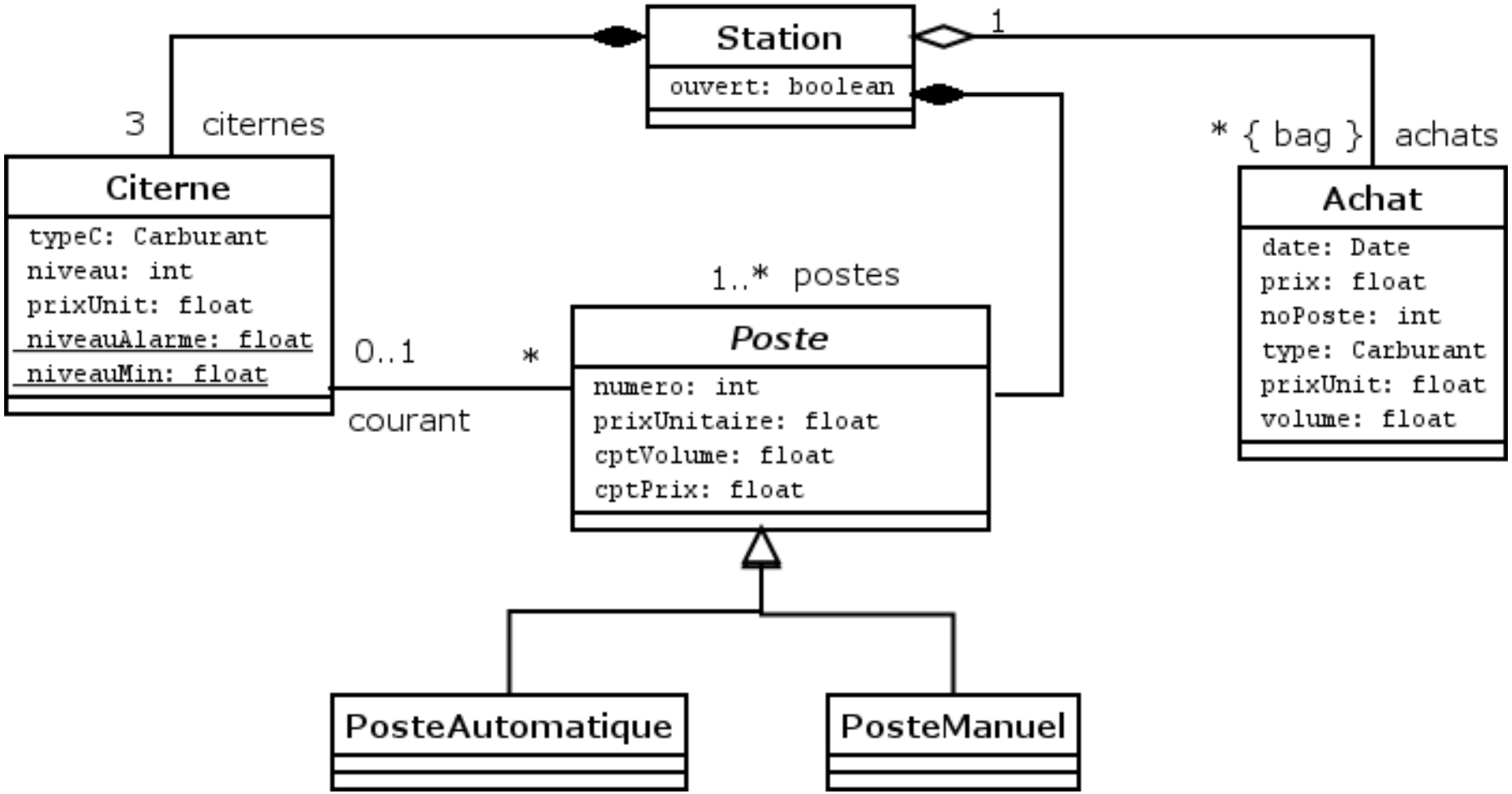


En mettant les variations d'association ensemble…



Inspiré de: « UML 2.0 Guide de référence », Rumbaugh et alli., CampusPress, 2005

Le diagramme de classe ATTOL final ?



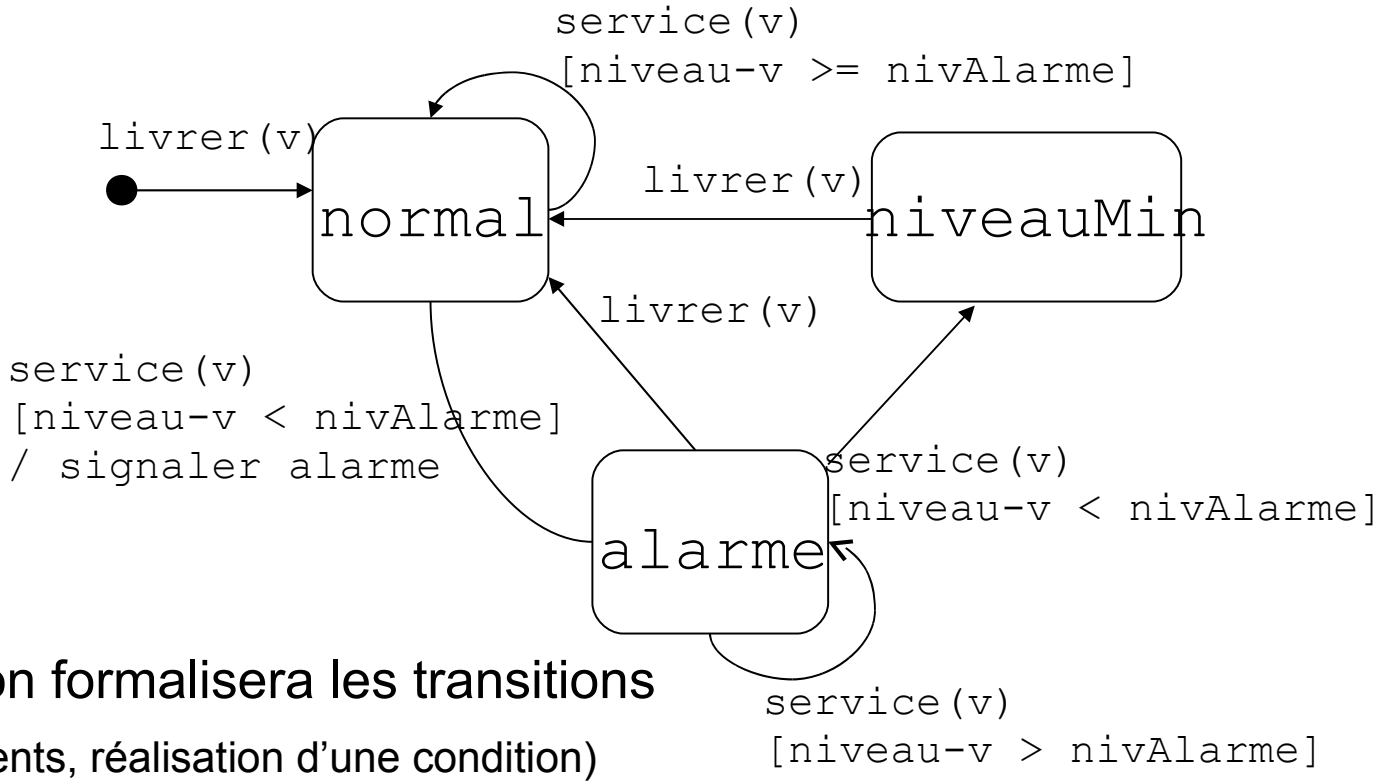
Exercice(s)…

- La station dispose de plus de 3 types de carburant, mais un poste n'offre toujours que 3 carburants.
- Historiser les achats: on distingue les achats consultables en ligne des achats archivés. A la fermeture de la station les achats du jour sont archivés.
- La station a un programme de fidélisation des clients (via une carte) qui leur donne des réductions et un ensemble de stations partenaires qui acceptent la même carte.
- Les tarifs peuvent varier (même dans la journée). On veut pouvoir retrouver le tarif en vigueur pour un achat.

☞ *Adaptez le diagramme de classes en conséquence…*

Une machine à états pour ATTOL

On veut insister sur la gestion des niveaux de carburant…



- Ultérieurement, on formalisera les transitions

  (méthodes, évènements, réalisation d'une condition)

- Donne une vue différente d'**une** instance (prototypique) de la classe, qu'on devrait retrouver implicitement ou explicitement dans l'impl …

En guise de synthèse pour l'analyse

En préalable à l'analyse

- Partir d'une première version de **l'analyse des besoins** pour aboutir à une première description du système
  - Être raisonnablement sceptique sur les termes utilisés: leur signification pour le client n'est pas forcément la même !
    - Si besoin, les ré-exprimer différemment, les illustrer par des situations
    - Si un terme technique est utilisé, expliciter les implications, les hypothèses simplificatrices qu'il amène ou les contraintes supplémentaires
    - Si besoin, faire un glossaire des termes techniques

En préalable à l'analyse (2)

- Établir clairement ce qui est dans/en-dehors du système
- Établir clairement les contraintes techniques :
  niveau de performance, contraintes de portabilité,
  environnement matériel et logiciel, type d'interface, …
- Établir les cas d'utilisations « non-fonctionnels »
  (sérialisation, archivage, redondance, module d'échanges
  avec d'autres applications)

☞ *Le document doit être validé par le client*

En préalable à l'analyse (3)

- Le client est peu disponible, et n'est pas un « pote »
  - Nommer un responsable des interactions avec le client
  - Préparer les réunions (documents, questions, déroulement)
  - Chaque réunion doit faire l'objet d'un compte-rendu
  - Les comptes-rendus sont diffusés et archivés
- Le client n'est pas forcément de votre domaine
  - Il n'anticipe pas forcément les limitations techniques, les implications en termes de délai, de coût, de technologie
  - Il connaît bien son domaine et oublie que vous ne le connaissez pas
  - Il y a des « non-dits », des aspects implicites (« évidences »)
  - Il ne connaît pas forcément exactement ses besoins, ou la façon de les exprimer clairement

Et réciproquement de votre part …

- Vous avez une obligation de conseil et d'écoute critique !
- Il ne parle pas forcément UML: du texte ? Des maquettes ?

Le processus d'analyse

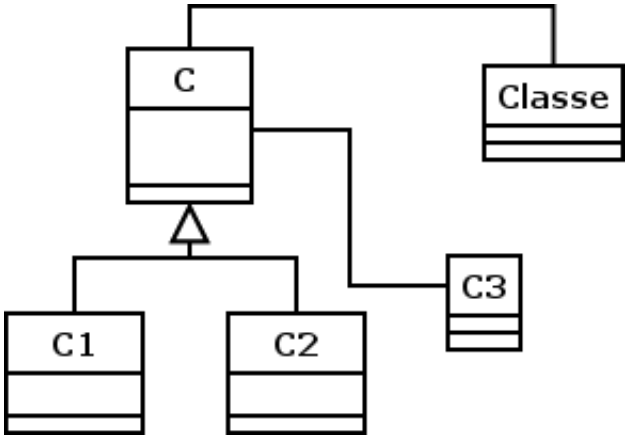
- Identifier les cas d'utilisation potentiels, les acteurs associés
- Produire un premier « *diagramme des classes du domaine* » en privilégiant classes élémentaires et associations
  - Identifier les termes du domaine (vocabulaire ou « abstractions » du monde modélisé) :
    - noms communs, personnes, qualificatifs, attributs, propriétés
  - Identifier les « responsabilités » associées à ces termes
  - Identifier qui est « moteur », qui est « sujet »
  - Identifier
    qui a besoin de « communiquer » avec qui,
    qui « partage de l'information » avec qui,
    qui est « en première ligne », qui n'est qu'un support …

Processus d'analyse (2)

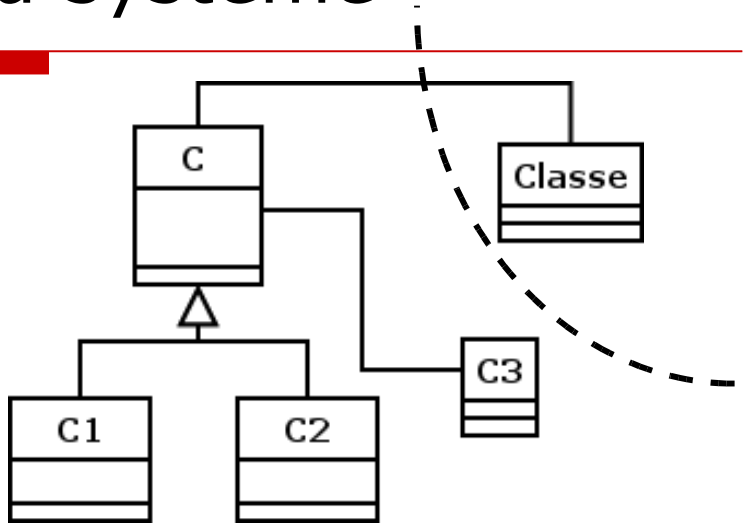
Raffiner ce premier diagramme

- Équilibrer les responsabilités entre abstractions (☹ une classe qui fait tout !)
- Éviter la dilution des responsabilités (regrouper les classes séparées mais inter-dépendantes)
- Distinguer entre abstractions et propriétés de ces abstractions (attributs)
- Distinguer les classes sans opérations naturelles (ce sont des types, pas des classes)
- Tracer les relations entre ces abstractions

Délimiter la frontière du système



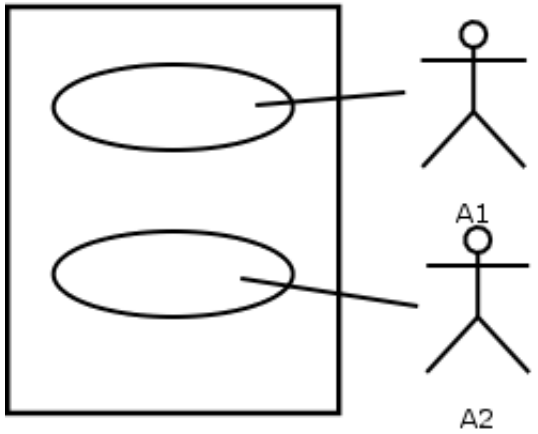
DCD: le domaine



DCS: le système

Déterminer ce qui dans le système

- En déduire les acteurs
- Faut-il une représentation des acteurs dans le système ?
- Mettre en évidence les interactions entre les acteurs et le système



Établir le premier modèle d'analyse

- Écrire le diagramme des cas d'utilisation
- Établir le modèle des classes
- Établir des scénarios d'utilisation
- Exprimer les invariants, les modèles d'opérations
- Déterminer si on a les bons « observateurs »
- Dériver des tests d'acceptation des scénarios: en parallèle avec l'analyse, on prépare les tests fonctionnels !
  - Quels enchaînements d'opérations à tester ?
  - Quels paramètres
  - Quelles instances pour leur exécution
  - Comment les mettre dans le bon état ?
  - Comment tester le résultat de l'enchaînement des opérations et l'état final du système

Environnement de test

Interface du système avec l'extérieur

- Introduire des classes d'interface (« classes actives ») pour gérer la communication avec les acteurs
- Comment passer d'une représentation externe à l'instance qui lui correspond dans le système ?
  - Le système d'information de la Banque connaît des « Clients »
  - Pour entrer dans le système un client s'identifie grâce à des caractéristiques externes (nom, numéro de compte, cookie, …)
- Avec les acteurs on n'échange généralement pas des objets puisqu'ils sont à l'extérieur du système !
- Comment passe-t-on du nom à l'instance ?
  - Table de hachage: nom -> Client ?
  - « mapping » relationnel + clef primaire ?
  - Collection statique de tous les clients + méthode de recherche ?
- Ces classes seront affinées lors de la conception !

Établir le « dictionnaire des données »

- Définit les identificateurs dans leur contexte
- En donne une description synthétique et claire
- Initié pendant l'analyse et enrichi ensuite

Nom	Catégorie	Description	Contexte
Poste	classe	texte court et explicatif !	n° § ou n°page
…	type		
…	acteur		
no	attribut		
achats	rôle		
…	…		

Documents à l'issue de l'analyse

- Cas d'utilisations, avec leurs spécifications (textuelles et sous forme de diagrammes de séquence)
- Scénarios qui illustrent la dynamique du système global
- Modèles des classes et invariants associés
- Modèles des principales opérations des cas d'utilisation
- Tests d'acceptation déduits des cas d'utilisation et des scénarios
- Dictionnaire des données
- Une version préliminaire du manuel utilisateur

Format des Documents

- Tout document doit être paginé et muni d'une table des matières
- Tout document doit être daté, muni d'un numéro de version, avoir une indication du rédacteur responsable
- Une page de garde doit résumer les principaux points modifiés par rapport à la version précédente (« historique » du document)
- Les versions doivent être archivées soigneusement: l'équipe doit conserver un exemplaire de chaque version.
- Il est conseillé d'effectuer des lectures croisées et « sceptiques » des documents
- Il y a un auteur mais l'équipe est collectivement responsable.

Compléments sur les machines à états

- Elles permettent de donner une autre vision des classes : comportement dynamique d'**une** instance donnée…
  - Dans quel état une méthode est-elle appelable ?
  - Quel est son effet, dans quel état laisse-t-elle l'instance ?
  - Quelles sont toutes les méthodes qui peuvent amener une instance dans un état donné ?

  Permet de vérifier qu'on oublie pas de transitions ou d'états !
- Les états peuvent rester implicites dans chaque méthode…
- … ou au contraire, on peut programmer l'instance comme un automate
  - `switch` à la Java/C, organisé autour d'une variable qui mémorise l'état courant
- A mi-chemin entre l'analyse et la conception !

Les machines à états

- Automates perfectionnés: états, transitions + « gardes », actions
- État: abstraction de la vie d'un objet pendant laquelle il satisfait une condition, exécute certaines activités et répond à des évènements extérieurs d'une certaine façon.
  - Notion d'état complexe (structuré)
  - Activités internes dans les états
  - Activités à l'entrée et à la sortie d'un état
- Transitions: relation entre deux états, reflétant généralement un changement de comportement.

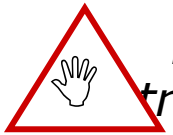
  Décrit par:
  - les états de départ et d'arrivée
  - l'évènement déclencheur
  - condition de franchissement (garde)
  - action(s) associée(s) à la transition

Les machines à états (2)

- Évènements : « influence » venue de l'extérieur pouvant induire une action de la part de l'instance
  - Appel (synchrone) d'une méthode de l'objet: `op(arg:T)`
  - Changement dans l'état d'une condition booléenne: `when(exp)`
  - Réception d'une communication asynchrone (signal): `sname(a:T)`
  - Écoulement d'une durée ou d'un temps absolu: `after(time)`
- Actions associées à une transition:
  - Action associée à la sortie de l'état courant (`exit`)
  - Action spécifiée sur la transition
  - Action associée à l'entrée dans l'état courant (`entry`)
  - Activité interne à l'état (`do`)

  « action » : traitement atomique

  « activité » : liste d'actions et/ou sous-activités



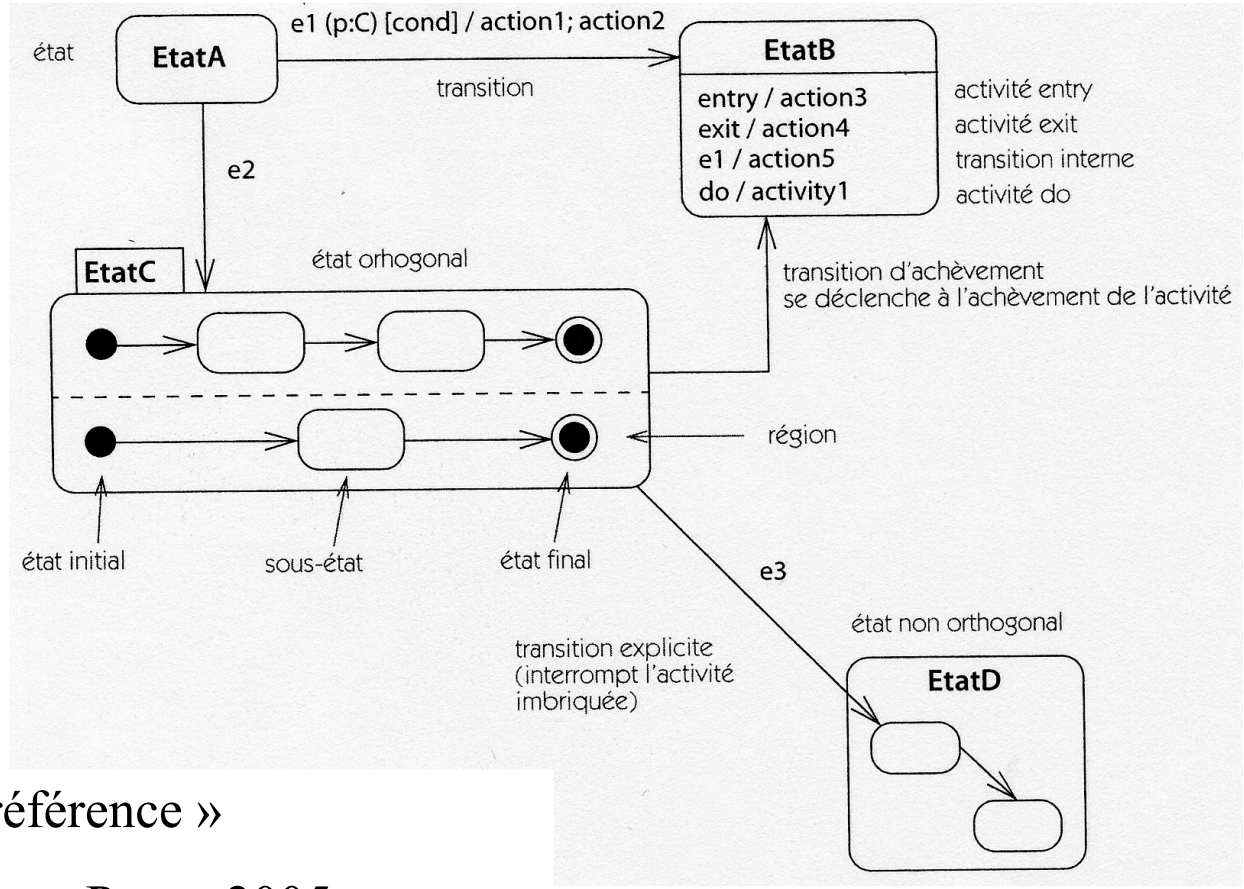
Pendant le traitement d'une transition, on ne peut pas traiter un autre évènement…

Les machines à états (3)

- Actions associées à une transition
  - Appel d'une méthode sur une instance
  - Modification d'un attribut
  - Envoi d'un signal à une instance
  - Création ou destruction d'un instance
  - …
- Possibilité de « transitions internes »
- États structurés: on peut décomposer en
  - États orthogonaux (parallèles ↔ facettes indépendantes)
  - États non orthogonaux (séquence de sous-états)
  - Transitions possibles à partir de l'état global et/ou des sous-états.

  ☞ *Sémantique complexe, difficile à maîtriser !*

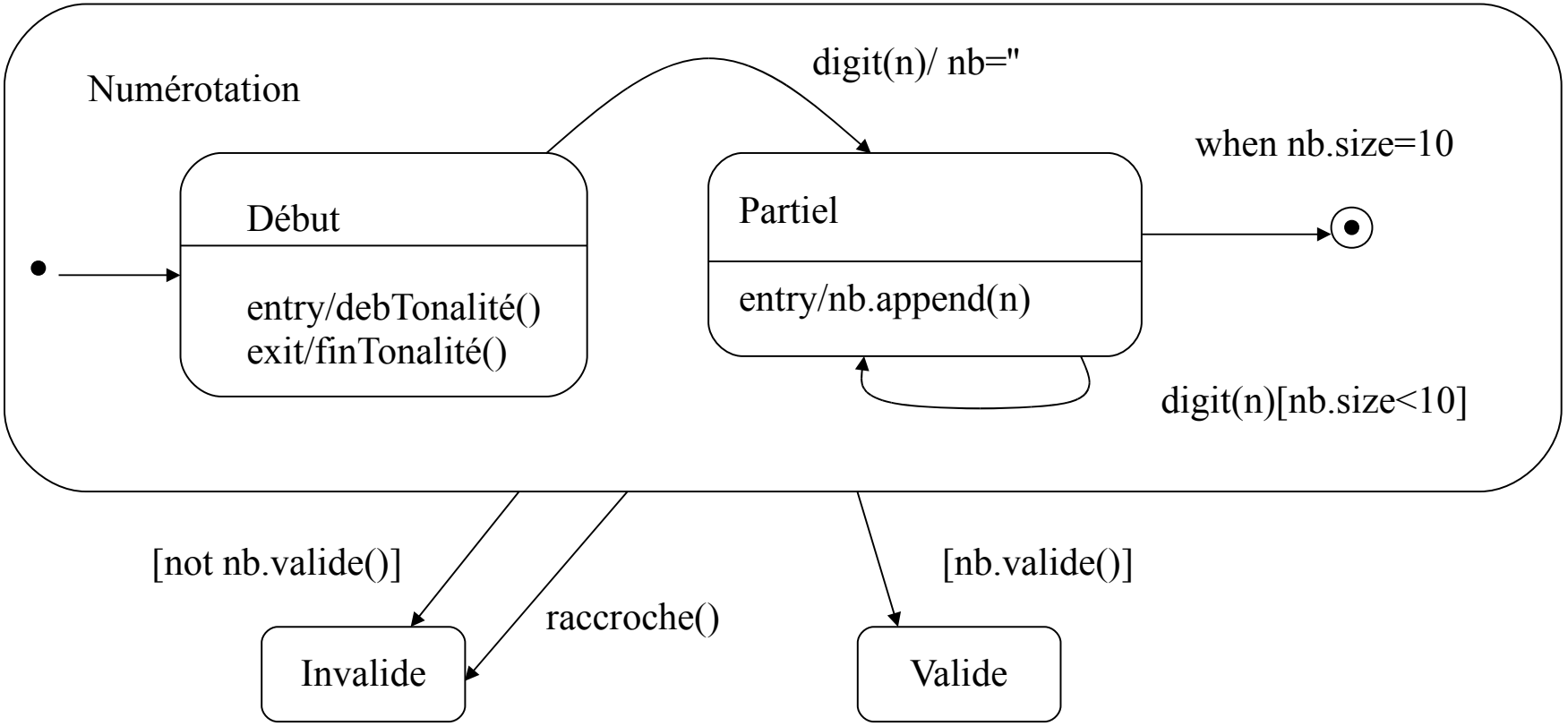
Machines à états : syntaxe étendue



« UML 2.0, Guide de référence »

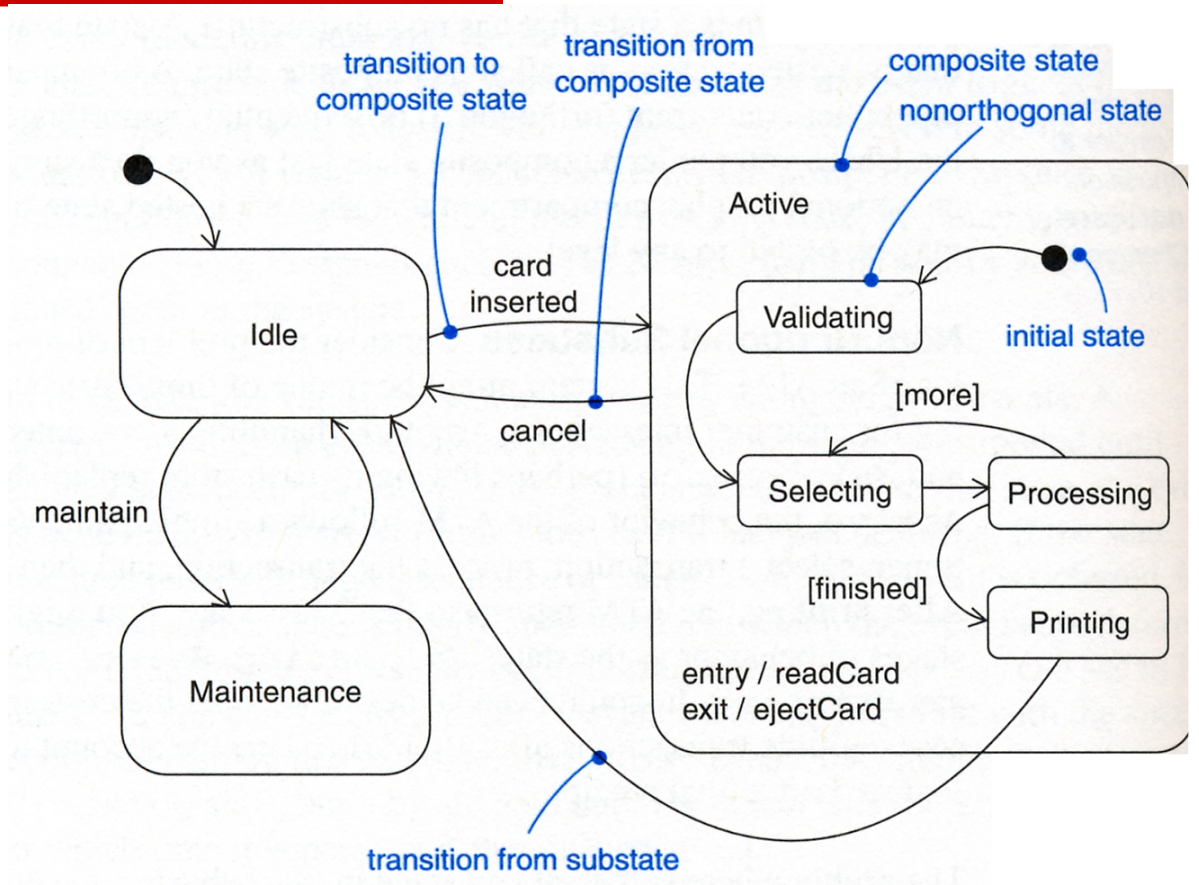
Rumbaugh & alli, CampusPress, 2005

Machines à états : exemple d'un téléphone



Exercice: modifier la machine à états pour accepter les numéros courts !

Machines à états : exemple d'un DAB



« The Unified Modeling Language User Guide »
G. Booch & alli, Addison-Wesley 2005